Mapping for All

Peter Mawhorter, Elaine Shaver, Zeke Koziol, and Zachary Dodds

Harvey Mudd College Computer Science Department
301 Platt Boulevard
Claremont, CA 91711
pmawhorter@cs.hmc.edu, eshaver, zkoziol, dodds@hmc.edu

Abstract

This synopsis presents Harvey Mudd College's entry into the 2008 AAAI robotics exhibition. We demonstrated three inexpensive robot platforms on which we have implemented a vision-based FastSLAM algorithm for landmark mapping. As a result, this project reinforces the *educational* scalability of low-cost platforms: they offer not only a compelling invitation to the study of computer science and robotics, but a means of engaging students in the recent and ongoing work of the AI robotics community.

Overview

In the 2008 AAAI robot exhibition Harvey Mudd College demonstrated several robotic platforms based on low-cost hardware. Each serves as a platform for running FastSLAM, a landmark-based mapping algorithm (Montemerlo et. al. 2002). Our goal was to demonstrate that inexpensive platforms, typically used to engage introductory students in computer science and/or robotics, can also serve as a basis for student investigations of recent and ongoing work in the field of AI robotics.

Platforms

Our first platform consists of an iRobot Create base connected directly to a MacBook Pro resting on top. For sensing, we attached an iSight webcam to the assembly. Not counting the laptop, the robot cost less than $400, putting it well within the means of many undergraduate programs. Since the laptop need not be dedicated to the robot – indeed, the laptop depicted is used for many other purposes – this platform meets our goals of affordability.

Also, because each of the components in Figure 1 is a commercial product, there is no special expertise or equipment required to build the robot from the parts shown. All that is necessary is some way to secure the laptop and webcam to the Create, a job for which velcro tape works more than well enough.

The use of the high-quality iSight camera makes this platform the most visually capable of our robots. Vision is the crucial sensing modality for the inexpensive and repeatable landmark-detection at the heart of the FastSLAM algorithm. Another advantage of this platform is the simplicity of its software interface. The Create has a standard serial API; the webcam interacts easily with the laptop, and can be accessed via the OpenCV library. This means that we didn't have to spend time writing low-level drivers for various components and could focus on the higher-level algorithmic concerns. This also means that the laptop must travel onboard; the Create is about as small as it can be while still handling such a payload.

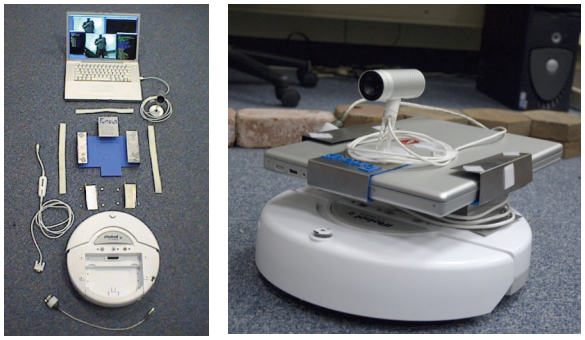

Figure 1 Our Create-based platform. At left is an image of the components comprising the robot; at right is the assembled platform. Beyond the existing laptop and webcam, which may be swapped as needed, the system's cost totals less than $200.

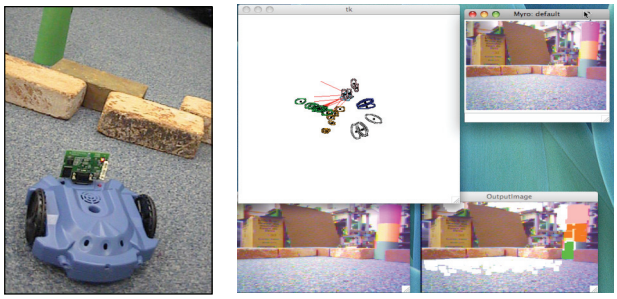

Figure 2 (left) The Myro platform consisting of a Scribbler base and Fluke Bluetooth camera. (right) The Scribbler's OpenCV-based vision system and FastSLAM mapping interface.

In addition to the Create-based platform, we implemented FastSLAM atop the Institute for Personal Robotics in Education's *Myro* hardware, consisting of a Parallax Scribbler base and a *Fluke* bluetooth camera. Transmitting images from the Fluke slows down the running of the algorithm, but it has the advantage of allowing students work entirely from a familiar laptop or desktop interface off-board the robot. Figure 2 illustrates these systems.

The FastSLAM Algorithm

Introduced in 2002 (Montemerlo et. al. 2002) and actively refined since that time, e.g., (Thrun et. al. 2004), FastSLAM combines both nonparametric and Gaussian representations of uncertainty in order to build and maintain a population of possible maps of a previously unknown environment. The algorithm processes a series of odometric estimates along with noisy landmark observations to produce an estimate of both robot location and landmark locations based on a combination of Kalman filtering and particle filtering. Although open-source implementations exist, we found that the community's current codebase serves researchers far better than it serves students. This is natural for an algorithm that continues to motivate advances in its field, but was not in line with our desire to make our code easy to use and understand.

To provide as accessible an implementation as possible, we implement the algorithm as a set of Python modules. Our goal in re-implementing the algorithm was to produce a well-commented piece of code that could be shared with other students or with researchers interested in *using*, rather than investigating, the approach. In addition, the use of Python dovetails with a strong codebase in place for both the Create and Myro hardware. We succeeded in running the resulting code on PC, Mac, and Linux-based systems without recourse to any third-party software libraries. Our FastSLAM codebase is freely available from `https://svn.cs.hmc.edu/svn/robotics/slam/pySLAM`.

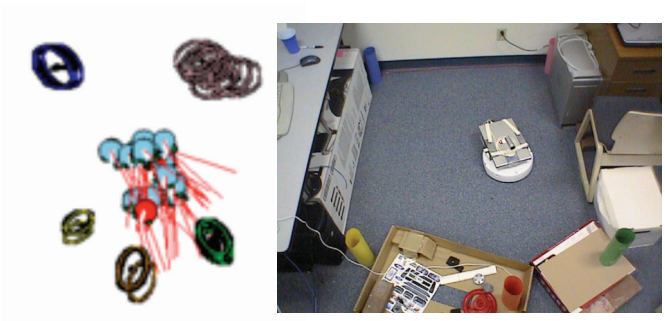

Figure 3 A population of maps is maintained by FastSLAM, at left. The red circle is the pose estimate based only on odometric data, showing the robot pointing to the side of the orange landmark. Cyan particles capture the uncertainty in the robot's location with a mode at the true pose, shown at right. The graphic at left superimposes the distinct map maintained by each pose.

The FastSLAM algorithm has several variants: FastSLAM 1.0, the algorithm that we have implemented in Python, is the simplest and most straightforward. FastSLAM 2.0 uses data about the observed landmarks not only to filter robot pose particles and to update landmark position estimates, but also to generate a more accurate set of estimated robot poses with each update. There are also several ways to run FastSLAM without knowing which landmark is which. This additional challenge requires a data-association metric to guide the algorithm to make choices about landmark correspondence. We have explored FastSLAM with unknown data association. However, our experiences suggest that distinguishable landmarks, e.g., the colorful construction paper cylinders shown in the figure, create a much more accessible – and successful – basis for initial student exploration of the approach.

The Vision System

The FastSLAM algorithm relies on measurements of odometry and landmark locations in order to compute its map. Rough odometry is straightforward to obtain. The Create platform has built-in wheel encoders; the Scribbler does not, but estimation by integrating the commanded velocities suffices. Reliably observing landmarks, on the other hand, poses a significant challenge. Many FastSLAM implementations segment landmarks from laser range-finder data, e.g., finding tree trunks or architectural corner features. To make FastSLAM more accessible on a smaller budget, we decided to use webcams to extract landmarks. Even with the straightforward color-segmentation of uniform cylindrical landmarks, our efforts with the vision system dominated the time required to implement the mapping algorithm.

Figure 4 Cylindrical landmarks have roughly constant width from any viewing angle. As a result, an estimated landmark distance can be computed from observed width.

Our vision algorithm is based on several simplifying assumptions. First, we assume that the landmarks that we are using are each uniquely-colored with respect to both each other and their surroundings. This enabled us to use relatively simple color-segmentation and blob-filling

methods to decide which part of an image contained a landmark and which landmark it was. Next, we assumed that our landmarks were perfect cylinders (and constructed them as close as possible to this ideal). This simplified the use of a perspective camera model to determine the range and bearing to an observed landmark: the range to a cylindrical landmark depends on its observed width, and a landmark's angle relative to the robot depends on its horizontal location in the image. Both computations require knowledge of the camera's field-of-view, internal parameters, and the assumption that it remains at a fixed distance above the ground plane. We estimated the needed information by imaging a ruler at a known distance. The resulting rough estimates suffice, because FastSLAM permits tuning of the error estimates of the landmark-sensing model: we simply increased the sensor-error covariance to encompass inaccuracies in calibration.

Using OpenCV and the CVBlobs library, we set up a visual pipeline that took images from the webcam, filtered them based on predefined color definitions, grouped blobs of colored pixels together, and calculated the range and bearing to any blobs observed. To improve the blobbing step, we use a dilate-erode filter pair with a higher radius for the erosion than the dilation. This meant that our observed width would be slightly less than the actual width, but it greatly reduced spurious blob detections. We accounted for the reduced width in our distance calculation algorithm. Using this software, we could use our webcam-equipped robots to make landmark observations and thus run FastSLAM. Complete source of our vision code is at `https://svn.cs.hmc.edu/svn/robotics/cvserver`. Our software enables any robot equipped with a webcam that can run OpenCV to recognize simple cylindrical landmarks made out of brightly-colored material. This crucial step makes it much easier to run algorithms that depend on observing landmarks on cheap platforms.

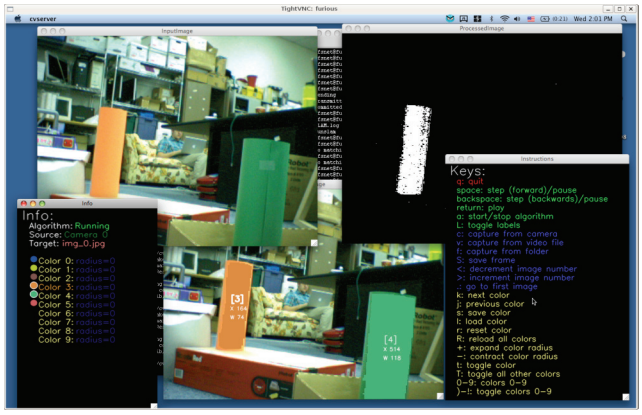

Figure 5 The vision algorithm processing an image. The window in the upper left is the raw image, the window in the upper right is the color segmentation, and the bottom image shows the blobs (represented by their average color) on top of the raw image, along with x-coordinate and width information.

Additional Platforms

Because one of the goals of our project was to be as inclusive as possible, we explored several platforms for use with our software. By running our software on several hardware setups, we were also able to verify the portability of our code and resolve several potential barriers to its wider adoption.

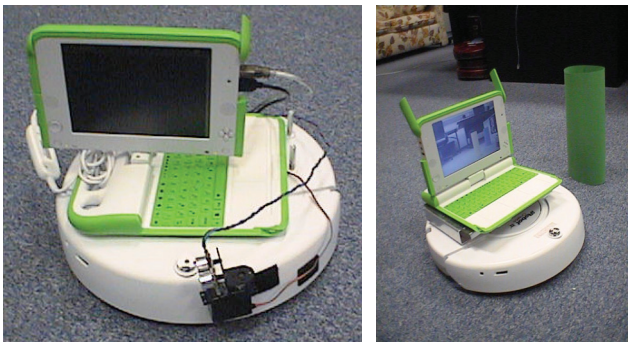

Figure 6 The OLPC laptop controlling an iRobot Create. This platform uses the OLPC's webcam for sensing as well as a sonar sensor mounted atop a servo motor on the front of the Create.

One platform that demonstrated our effort's portability is the One Laptop Per Child Foundation's XO laptop, also known as the "OLPC." Although no longer available commercially, this platform is not uncommon, having been mass-produced for deployment around the world. Our institution obtained ten OLPCs for a prior term's *Computing in the Developing World* seminar. As a result, we were able to leverage them as an alternative platform for driving the Create. The OLPC is superior to a commodity laptop both in durability and cost. The OLPC has a built-in webcam and supports OpenCV (albeit with some modifications). The resulting cost of Figure 6's vision-based robotic system *including all computation and development systems* is less than $400, far less than other vision-based robots' price points. As a result, we term this system the *ORPC*, or One Robot Per Child platform.

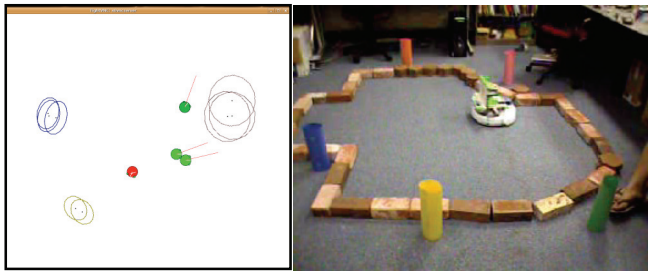

Figure 7 (left) A snapshot of FastSLAM's state. Three (green) robot-pose particles and their maps are juxtaposed with the odometric position in red. The actual location of the ORPC appears at right. Note that the camera is positioned to observe its laptop's user, so that it is facing leftward. Only the pink blue and yellow landmarks have been observed thus far. The ellipses show their means and covariances in each of the maps maintained.

The ORPC's primary drawback is its custom Linux-based operating system, whose stringent security limitations intend to protect users new to computation. Those protections can complicate the creation and composition of systems that connect the OLPC's hardware and software. After finding several workarounds for components that behaved differently on the OLPC – and modifying our code to be more portable in the process – we were able to get our FastSLAM implementation working on our ORPC platform. The main hitches were with the camera, for which we had to use a custom-compiled version of OpenCV, and the access protections. Our repository, noted above, contains the final version of our ORPC code.

A second platform we considered was the Qwerkbot, a robot based on the work of CMU's TeRK project. This robot is built around an embedded Linux board, called the *Qwerk*, which offers a capable computational engine and many input/output ports. We attached a pair of wheels, a webcam, and 5 sonars to the Qwerk. To facilitate development, the robot sends the video stream offboard as a sequence of pictures to be processed by a remote laptop using an ad-hoc wireless network. This effort ran into problems with the Logitech webcam. In particular its white balance aggressively adjusted to the lighting conditions, washing out landmarks as shown in Figure 8 and foiling color-region extraction algorithms.

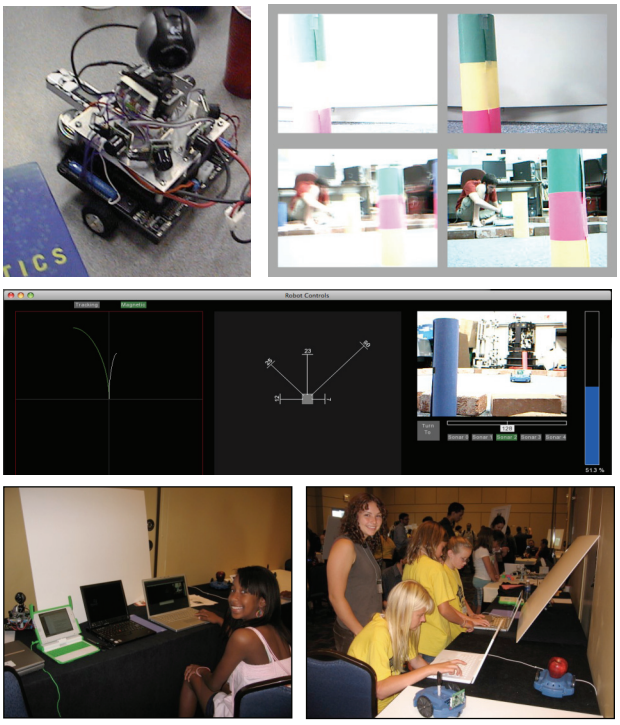

Figure 8 (top left) The Qwerkbot platform. (top right) An image demonstrating the overly aggressive white-balancing of the Qwerk board's camera drivers. (middle) The robot's interface has velocity control, camera panning, sonar data, and a video feed. (bottom) Middle-school girls visiting AAAI 2008 found the tabletop platforms engaging, challenging, and accessible.

Faced with this failure we have moved towards SIFT-based landmarks, and progress continues in the creation of that system.

Results and Perspective

In the end, the exhibition at AAAI 2008 offered precisely the counterpoints this project sought. For most of the three day event, our platforms provided grist for AI and robotics educators: discussions focused on the accessibility of spatial reasoning algorithms at the undergraduate level. Yet for two hours on the final day, the character of the exhibition changed completely. A troop of three dozen middle-school girls visited the AAAI's robot exhibition as part of their science-themed summer camp (Figure 8).

During this visit, the two tabletop platforms – the Qwerkbot and the Scribbler – showed their remarkable pedagogical range. The girls engaged in hands-on challenges in which they controlled those robots to accomplish small, challenging tasks. One of these tasks asked the campers to program the Scribbler to trace the first letter of their name; in another challenge, girls used only the sensor data available to the Qwerkbot in order to direct it safely though an unknown and otherwise unseen enclosure.

By providing the capability to support FastSLAM on robots accessible enough to actively engage middle-school students in open-ended investigations, these platforms offer a baseline of *pedagogical scalability* that we hope to continue to improve. Our ongoing efforts seek to implement additional spatial-reasoning and vision algorithms (SIFT and monocular depth) that will make inexpensive robots even more compelling resources for educators at a wide range of curricular levels.

Acknowledgments

The authors gratefully acknowledge support from National Science Foundation DUE CCLI #0536173, the Institute for Personal Robotics in Education, and resources provided by Harvey Mudd College.

References

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit. B. (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem, *Proceedings of the National Conference on Artificial Intelligence*, pp. 593-598.

Thrun, S., Montemerlo, M., Koller, D., Wegbreit, B., Nieto, J., and Nebot, E. (2004). FastSLAM: An Efficient Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association, *Journal of Machine Learning Research,* 4(3) pp. 380-407.